

Building Resource Adaptive Software Systems (BRASS): Objectives and System Evaluation

Jeffrey Hughes¹, Cassandra Sparks¹, Alley Stoughton^{1,†}, Rinku Parikh², Albert Reuther¹, and Suresh Jagannathan²

¹MIT Lincoln Laboratory
Lexington, MA USA 02421

²Defense Advanced Research Projects Agency
Arlington, VA USA 22203

Abstract—

As modern software systems continue inexorably to increase in complexity and capability, users have become accustomed to periodic cycles of updating and upgrading to avoid obsolescence—if at some cost in terms of frustration. In the case of the U.S. military, having access to well-functioning software systems and underlying content is critical to national security, but updates are no less problematic than among civilian users and often demand considerable time and expense. To address these challenges, DARPA has announced a new four-year research project to investigate the fundamental computational and algorithmic requirements necessary for software systems and data to remain robust and functional in excess of 100 years. The Building Resource Adaptive Software Systems, or BRASS, program seeks to realize foundational advances in the design and implementation of long-lived software systems that can dynamically adapt to changes in the resources they depend upon and environments in which they operate. [1] MIT Lincoln Laboratory will provide the test framework and evaluation of proposed software tools in support of this revolutionary vision.

I. BACKGROUND

Modern-day software systems operate within a complex ecosystem comprising a myriad of sophisticated libraries and middleware, managed services, protocols, models, drivers, browsers, storage systems, databases, etc. To manage this complexity, applications typically bake-in many assumptions about the expected behavior of their ecosystem; these assumptions, unfortunately, may be implicitly violated as the ecosystem changes. Advances in technology result in platform upgrades with new processors, devices, or architectures, which may be defined in terms of significantly different data formats, new programming paradigms, and dramatically different object, programming, or memory models. Vulnerabilities or deficiencies in system and library code (security, logical, or performance related) can result in substantial modification to, and refactoring of, existing scripts, APIs, data representations, and protocols. The structure of application inputs may vary over time, exercising parts of the ecosystem that were previously not considered, or exercising previously studied components in new and unexpected ways. Long-lived, mission-critical systems operate using evolving and reconfigurable system

resources in which availability constraints are dictated by varying mission-parameter demands.

Ensuring applications continue to function as expected in the face of such a changing operational environment is thus a formidable challenge. Failure to effectively and timely respond to these changes can result in technically inferior and potentially vulnerable systems, but, the lack of automated mechanisms to restructure and transform applications when changes do happen, leads to high software maintenance costs and premature obsolescence of otherwise functionally sound systems. Consequently, the inability to seamlessly adapt to new operating conditions negatively impacts economic productivity, hampers the development of resilient and secure cyber-infrastructure, raises the long-term risk that access to important digital content will be lost as the software that generates and interprets that content becomes outdated, and complicates the construction of autonomous mission-critical programs. [2]

Precisely understanding application intent, a characterization of an application that extends beyond just its functional specification, but additionally subsumes other algorithmic characteristics such as the shape and structure of its inputs, notions of performance and efficiency, reliability, fault-tolerance, choice of data representations, cost models, security policies, etc., is fundamental to addressing this challenge. While implementations often closely interact with their environment during development to ensure that desired intent is preserved, whatever guarantees are derived during that process often do not hold in the face of arbitrary (continuous or discrete) ecosystem evolution, which can manifest in a number of different ways:

- **Restricted functionality** – As ecosystems evolve, they may limit how resources such as memory, communication bandwidth, energy, and processing power are apportioned and leveraged to reflect specific characteristics of new hardware, new programming paradigms, or new end-user requirements.
- **Enhanced functionality** – The set of protocols and services provided by an ecosystem may change over time to include additional features and functionality not originally available.
- **Altered functionality** – Changes in data formats, protocols, input characteristics, and models (e.g., programming or memory) of components in a software ecosystem may violate or undermine various (often implicit) application assumptions.

*This work is sponsored by Defense Advanced Research Projects Agency (DARPA) under Air Force contract FA8702-15-D-0001. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government. Approved for public release: distribution is unlimited.

[†]Now affiliated with IMDEA Software Institute

Adapting applications to effectively operate within an evolving ecosystem thus necessitates the ability to infer and discover the impact of environment changes on application behavior and performance, and transform applications to beneficially and safely exploit these changes, with the expectation that any transformations performed (even if they change intent) still guarantee high fidelity of the transformed application with respect to the functional characteristics of the original. [3]

II. PROGRAM OVERVIEW

The goal of the BRASS program is to realize foundational advances in the design and implementation of survivable, long-lived complex software systems that are robust to changes in the resources (logical or physical) provided by their ecosystem. These advances will necessitate (1) integration of new linguistic abstractions, scalable and compositional formal methods, resource-aware program analyses, and related approaches to discover and specify application intent, and (2) novel runtime and operating system designs, along with program transformation and patching strategies to guarantee and provide high-assurance validation of important application invariants that are preserved in the face of an evolving underlying ecosystem. Towards this end, BRASS seeks new approaches to enable automated discovery of relationships between computations and the resources they utilize, along with new techniques to safely and dynamically incorporate optimized algorithms and implementations constructed in response to ecosystem changes.

It is envisioned that a system capable of accomplishing these goals will, at a minimum, be composed of the following three technical areas (TA): TA1 Platform, TA2 Analytics and TA3 Discovery. The role of each of these will be covered in the following subsections.

A. TA1: Platform

The goal of TA1: Platform is to explore the applicability of adaptive and transformation analysis techniques to the development of resource adaptive systems, using the proposed platforms selected for case studies. Exemplar platforms include, but are not limited to: autonomous and robotic systems, embedded systems (e.g., Internet of Things), geo-distributed systems, heterogeneous scalable multiprocessors, cloud infrastructure, mobile platforms, high-assurance systems, coordinated platform ensembles, storage and file systems and diverse hardware infrastructures (e.g., FPGA, SoC).

B. TA2: Analytics

TA2: Analytics is responsible for monitoring system runtime behavior with respect to a given application or set of applications, identifying salient changes to the underlying system that may affect application intent, selecting transformations based on analyses provided by TA3 activities, and incorporating program transformations and restructuring techniques to evolve applications to operate effectively within the context of a new ecosystem. The scope and extent of these transformations very much depends upon the accuracy and sophistication of the

TA3 analyses, the ability of compilation and runtime systems to perform updates efficiently, and an understanding of the platform in which the ecosystem resides.

The monitoring component of the TA2 effort must guide transformation and selection strategies that propose alternative implementations for affected parts of an application. Among others, these implementations can be: (a) captured by different versions of an application component, (b) composed of different algorithmic variants that all share the same interface signatures, but have different underlying instantiations and complexity, tailored for different resource characteristics and inputs, or (c) discovered via relationships and ontologies built in a software mining repository. Regardless of the mechanisms used to identify suitable variants, the transformation component of this effort must stitch the chosen variant into the application, ensuring type correctness and safety of the resulting artifact.

C. TA3: Discovery

The goal of TA3: Discovery is to provide a detailed understanding of program properties with respect to the resources they depend on in their underlying ecosystems, sufficiently precise to enable subsequent transformation and adaptation. These properties may be defined, discovered, and validated though the use of program annotations and explicit specifications, program analysis, type systems, contract systems, aspect-oriented programming abstractions, test mechanisms, abstract interpretation, model checking, termination analysis, shape and heap analysis, worst-case execution time analysis, etc.

In addition, machine-learning approaches that discover and predict properties relating program behavior to ecosystem resources may also be applicable. The outputs of executable language models, language or systems-based provenance extraction, documentation analysis (e.g., natural language interpretation of source-code comments), and other less traditional mechanisms for deriving program properties are also valid avenues of study within this area.

III. SUMMARY

This talk will provide an overview of the BRASS program goals and methodology as well as a discussion of our work developing a software framework for ecosystem evolution and system monitoring used to evaluate technologies aimed at addressing this problem.

REFERENCES

- [1] S. Jagannathan, "DARPA Seeks to Create Software Systems That Could Last 100 Years," April 08, 2015. [Online]. Available: <http://www.darpa.mil/news-events/2015-04-08>
- [2] —, "Building Resource Adaptive Software Systems (BRASS) DARPA-SN-15-35," March 19, 2015, Proposers' Day Announcement. [Online]. Available: <https://www.fbo.gov/utills/view?id=d8b3ca3a22d4df46e4bb5796840d99f7>
- [3] —, "Building Resource Adaptive Software Systems (BRASS)," April 7, 2015, DARPA-BAA-15-36 BRASS. [Online]. Available: <https://www.fbo.gov/utills/view?id=8af18d5c0a0eda51027474c3005c7660>